2019-04-30

Automatic Generation of Contractual Requirements From MBSE Artifacts

Salado, Alejandro; Wach, Paul

Monterey, California. Naval Postgraduate School

SYM-AM-19-042

PROCEEDINGS
OF THE
SIXTEENTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM

WEDNESDAY SESSIONS
VOLUME I

Acquisition Research:
Creating Synergy for Informed Change

May 8–9, 2019

Published: April 30, 2019

Prepared for the Naval Postgraduate School, Monterey, CA 93943.

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

Automatic Generation of Contractual Requirements From MBSE Artifacts

Alejandro Salado—is an Assistant Professor with the Grado Department of Industrial and Systems Engineering at Virginia Tech. His research focuses on applying decision analysis to improve the practice of engineering, in particular in the areas of verification and validation, and on improving problem formulation through modeling. Dr. Salado is a recipient of the NSF CAREER Award and the Fulbright International Science and Technology Award. He holds a BSc and an MSc in electrical engineering (Polytechnic University of Valencia), an MSc in project management and an MSc in electronics engineering (Polytechnic University of Catalonia), the SpaceTech MEng in space systems engineering (Delft University of Technology), and a PhD in systems engineering (Stevens Institute of Technology). [asalado@vt.edu]

Paul Wach—is a PhD student in Systems Engineering at Virginia Tech. His research interests include the mathematical formalism of model-based systems engineering (MBSE). Wach is currently exploring the feasibility of underpinning the Systems Modeling Language (SysML) with the mathematical Wymorian System Construct. He is also employed by the Department of Energy (DOE), where he manages $4 billion of work. While at the DOE, Wach has led implementation of enterprise- and program-level systems engineering and program management practices. He has previously worked for two of the DOE national laboratories, Pacific Northwest National Laboratory and Savannah River National Laboratory. Prior to work with the DOE labs, Wach was developing cutting edge artificial kidney technology based on his Master of Science with the University of South Carolina and medical research experience at the Medical College of Georgia. He also holds a Bachelor of Science degree in Biomedical Engineering from the Georgia Tech. [paulw86@vt.edu]

Abstract

This paper is intended to disseminate initial outcomes of the NPS Research Acquisition Program "Automatic Generation of Contractual Requirements from MBSE Artifacts" project. The research addresses the automatic generation of contractual requirements in textual form from models in a Model-Based Systems Engineering (MBSE) environment, enabling the transition from document-centric systems engineering to MBSE in acquisition programs. Textual requirements form the backbone of contracting in acquisition programs. Requirements define the problem boundaries within which contractors try to find acceptable solutions (design systems). At the same time, requirements are the criteria by which a customer measures the extent to which their contract has been fulfilled by the contractor. However, latent problems exist in acquisition programs stemming from poor practices in requirements engineering. Research suggests that transitioning to model-based requirements can be effective in coping with such challenges. We presented in prior work a framework to construct true model-based requirements within the context of the Systems Modeling Language (SysML). This research addresses the main question of whether contractual requirements in textual form can be automatically generated from those requirement models without loss of information or intent. We present in this paper an initial template of requirements and a process to support this goal.

Introduction

Textual requirements form the backbone of contracting in acquisition programs. Requirements define the problem boundaries within which contractors try to find acceptable solutions (design systems; Salado et al., 2017). At the same time, requirements are the criteria by which a customer measures the extent to which its contract has been fulfilled by the contractor (e.g., INCOSE, 2015). Hence, it is not surprising that some authors consider requirements "the cornerstone of … systems engineering" (Buede, 2009). However,

literature shows latent problems in acquisition programs stemming from poor practices in requirements engineering (e.g., Yeo, 2002; Dada, 2006; McConnell, 2001; El Eman & Birk, 2000).

In order to cope with such a challenge, academia and industry envision extending the application of Model-Based Systems Engineering (MBSE) beyond conceptual design, particularly addressing problem formulation. Two main paths to integrate requirements within a complete MBSE environment are currently pursued. In the first path, major modeling languages, such as Systems Modeling Language (SysML), incorporate elements called *requirement models* (Friedenthal, Moore, & Steiner, 2015), which are intended to model the requirements the system is expected to fulfil. Some authors have attempted to demonstrate how those so-called *requirement models* can be used to move acquisition practice from document-centric (textual) requirements to model-based requirements (e.g., Holt et al., 2011; Holt et al., 2015). However, this approach is based on defining specific model elements, called "requirements," which contain a text property that takes the textual requirement. The requirement element is then linked to a specific component in the system architecture. Hence, the only modeling value of this approach is to achieve traceability between requirements and architectural elements. Although this is valuable on its own merit, requirements remain textual; thus, model-based requirements are not achieved.

In the second path, researchers propose to use behavioral models of the system of interest as problem definition elements (requirements; e.g., Miotto, 2014). Such work has been confined, though, to the technical challenges of modeling expected system behavior. Therefore, the proposition remains positional, since such work has not addressed how contracting in acquisition programs is affected, or needs to be adjusted, to incorporate behavioral models as a contractual mechanism instead of textual requirements. Hence, the near-term, practical feasibility of the approach is questionable.

In a third path, less extended, mathematical or formal structures are used to capture requirements (e.g., Micouin, 2008). In these approaches, *shall* statements or similar natural language statements are not used in the formulation of the requirement. In the context of the research presented in this paper, these representations may be considered examples of true model-based requirements. Their usage in the context of SysML is, however, not evident.

The overarching research in which this paper is framed is aimed at overcoming those obstacles by providing a translation mechanism that enables the engineering of true requirement models, while automatically generating corresponding textual requirements. Prior work by the authors has addressed the construction of such true model-based requirements in SysML (Salado & Wach, 2019). This paper presents a template and showcases a requirement translation process that enables the automatic generation of contractual requirements in natural language (i.e., textual requirements) from model-based requirements.

Background: Model-Based Requirements in SysML

The construct for model-based requirements in SysML described in Salado and Wach (2019) is used in this paper. A summary of the construction specification for such model-based requirements is provided in this section.

Justification

The key underlying construct of a model-based requirement lays upon "the central proposition … that every requirement can be modeled as an input/output transformation" executed through one or more physical interfaces (Salado & Wach, 2019). This proposition

is founded on two main premises. First, every system can be modeled as a transformation of input trajectories into output trajectories (Wymore, 1993). Second, a set of requirements yields a solution space (Salado, Nilchiani, & Verma, 2017). Therefore, "it follows that a solution space can be modeled as a set of transformations of input trajectories into output trajectories" (Salado & Wach, 2019).

The suitability of this construct was explored by re-interpreting requirement categories of a taxonomy that fulfills the partition criterion as input/output transformations (Salado & Wach, 2019). Four requirement types, which are considered to be collectively exhaustive to capture requirements, were considered: functional requirements (i.e., what the system must do), performance requirements (i.e., how well the system must do it), resource requirements (i.e., what the system may consume to do those things that well), and environmental requirements (i.e., in which settings or contexts the system must do those things, that well, with those resources; Salado & Nilchiani, 2014, 2017). The explanation of how these types of requirements may be described as sets of input/output transformations provided in Salado and Wach (2019) is reproduced verbatim here:

> *Functional requirements* inherently describe input/output transformations. Mathematically, a function is necessarily defined as a mapping between a domain and codomain. From a General Systems Theory perspective, engineered systems are necessarily open (von Bertalanffy, 1969).
>
> *Performance requirements* are, as defined, necessary characteristics, properties, or attributes associated with the inputs and outputs of the transformations that the system shall perform. In fact, this condition is necessary because any attribute transparent to the interaction between the system and external systems should not be considered a requirement due to unnecessarily constraining the solution space (Salado et al., 2017, INCOSE, 2012).
>
> *Resource requirements* define limits on resources that the system may consume. It is obvious that a resource must therefore be inputted to the system and that it is consumed for producing something. Hence, any limitation imposed on resource consumption is in fact part of a functional exchange and can be modeled in such a way.
>
> *An environment for the system* is an abstraction of boundaries between the system and external systems. The environment provides certain conditions under which the system must operate and imposes certain limitations on how the system may affect the environment. In other words, the environment provides certain *inputs* under which the system must operate and imposes certain limitations on the *outputs* the system may yield to the environment.

In terms of typology of inputs and outputs, the construct is consistent with Kossiakoff et al.'s (2011) taxonomy for external interfaces and considers that systems operate in three types of media (information, material, and energy) that become inputs to and/or outputs from the system (Salado & Wach, 2019).

Construction Rules

A complete description of the construction rules for the model-based requirements is given in Salado & Wach (2019). A summary is provided here.

In line with the theoretical construct described in the previous section, the model-based requirements are built according to the meta-model depicted in Figure 1.

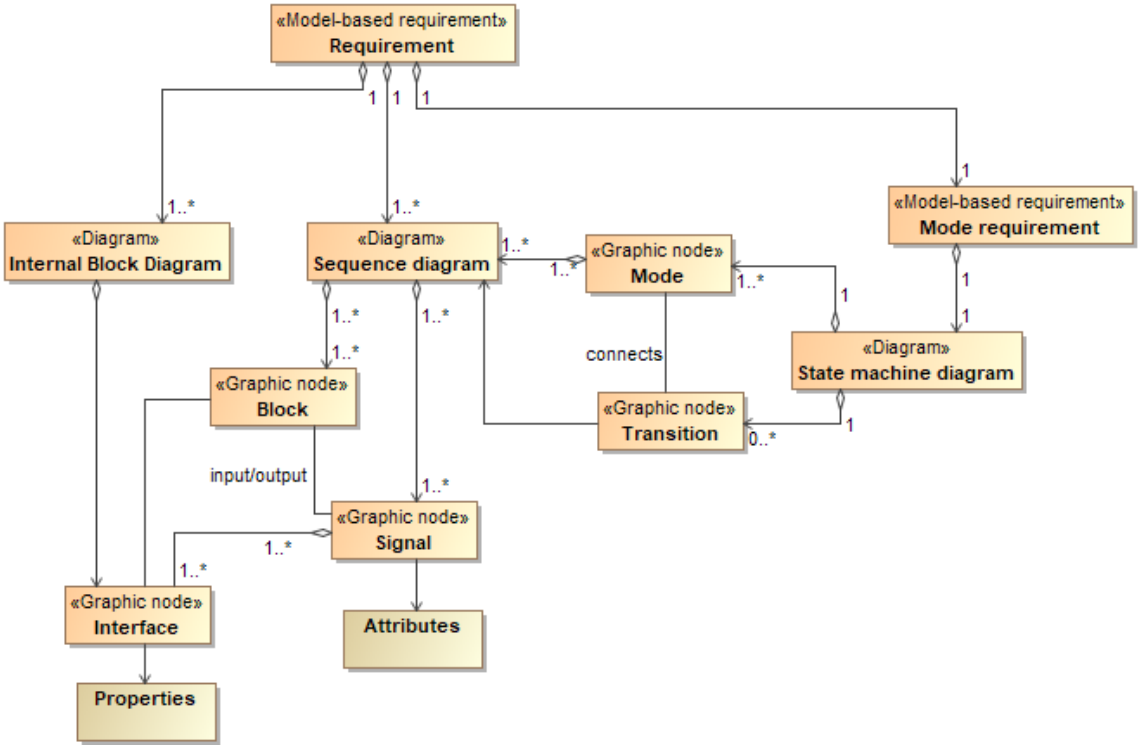


Figure 1. Meta-Model of the Model-Based Requirements
(Salado & Wach, 2019)

Three main SysML constructs are used to capture requirements as models (Salado & Wach, 2019):

1. A sequence diagram, which captures the required input/output exchanges. Each input or output is modeled by signal elements, which capture the required properties of each input and output. An example is provided in Figure 2.
2. An internal block diagram, which captures the physical interfaces that are required to convey the required system inputs and outputs. Each interface is modeled by ports, which capture the required properties of each interface and the signals it conveys. An example is provided in Figure 3.
3. Mode requirements, which describe the sets of requirements that apply simultaneously, modeled by state machine diagrams. Each state represents a mode, which represents a collection of requirements that need to be fulfilled simultaneously. An example is provided in Figure 4.

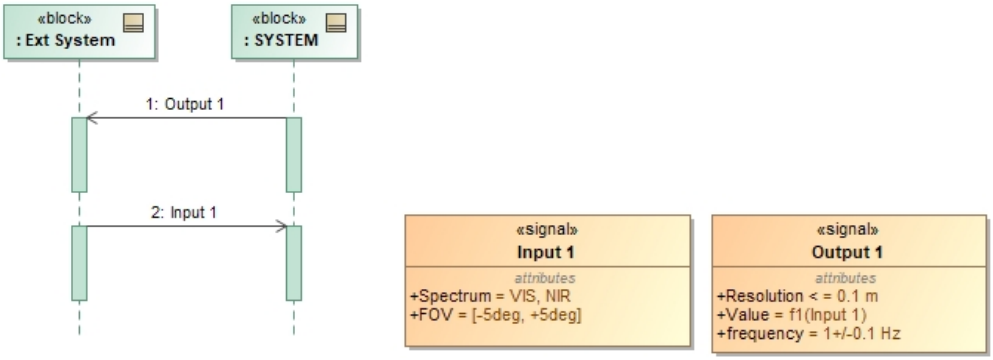


Figure 2. Example of Input/Output Transformation As a Model-Based Requirement
(Salado & Wach, 2019)

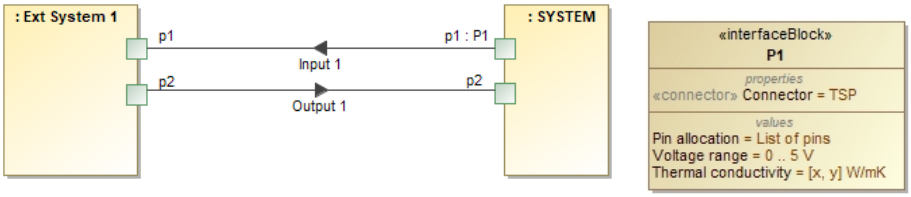


Figure 3. Example of a Required Physical Interface Through Which the Required Input/Output Transformation Occurs as a Model-Based Requirement
(Salado & Wach, 2019)

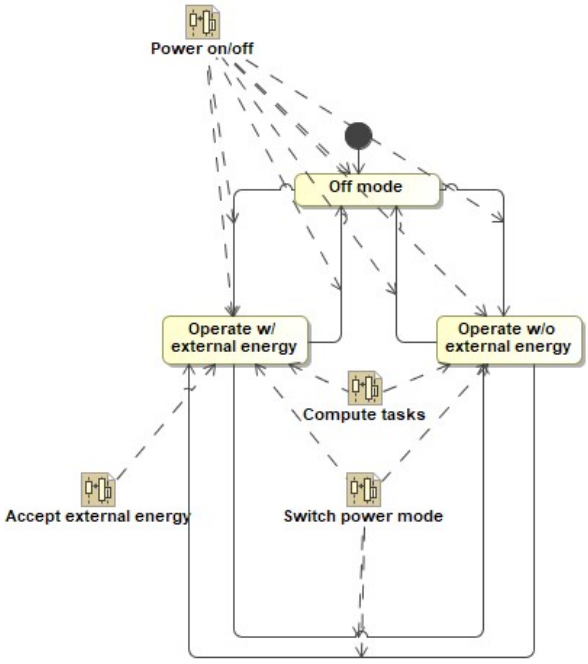


Figure 4. Example of Requirement Sets as a Model-Based Requirement
(Salado & Wach, 2019)

It should be noted that although existing SysML constructs are used to model requirements, there are semantic differences with respect to their regular use to model system solutions (Salado & Wach, 2019). Describing those differences is outside the scope of this paper because they are addressed in the original source. It suffices to state that the diagrams shown in this section extend (or modify in some cases) their traditional use in SysML. In essence, they should not be interpreted as models of the behavior or physical structure of the system, but as models of the input/output transformations the system is required to execute.

An Approach to Transform Model-Based Requirements to Contractual Requirements in Natural Language

Process

The process to transform the model-based requirements presented in the previous section (Background: Model-Based Requirements in SysML) to contractual requirements in natural language consists of four steps:

Step 1. For each port, generate corresponding textual requirements. This step generates a list of physical interfaces that are characterized by a set of required properties, which will be pointed at by the requirements resulting from the sequence diagrams.

Step 2. For each mode, generate a simultaneity modifier. This step assigns tags to each sequence diagram associated with a particular mode. These tags are used later to associate a modifier with the textual requirements resulting from such sequence diagrams that indicates the need to fulfill such requirements in the context of all other requirements with the same modifier.

Step 3. For each sequence diagram, generate corresponding textual requirements. This step generates a list that contains requirements associated with the need to accept inputs and provide outputs, the characteristics of those inputs and outputs, and the logical or temporal conditions for the acceptance of those inputs and provision of those outputs. In addition, for each requirement referring to the required inputs and outputs, a modifier referring to the physical interface through which such input or output is conveyed is added. Furthermore, the simultaneity modifiers in Step 2 are used to identify the subset of requirements that need to be fulfilled simultaneously.

Step 4. Remove repetitions, if any. Because inputs and outputs may be used in several sequence diagrams, this step will consolidate the list of requirements to avoid repetitions. It should be noted that this step can be executed after all textual requirements have been generated or as they are being generated, for efficiency purposes.

The basic concept for generating textual requirements leverages a predefined template of natural language requirements that maps to the different elements in the meta-model depicted in Figure 1. A simplified view of this concept is shown in Figure 5. A computerized algorithm is not used in this paper but is being developed as part of the research program. It will be disseminated in future publications. The focus of this paper lays on the template that will be employed to generate the textual requirements. Specific template rules are defined, as will be described in the next section, to cope with the different types of requirements captured by the model-based requirements.

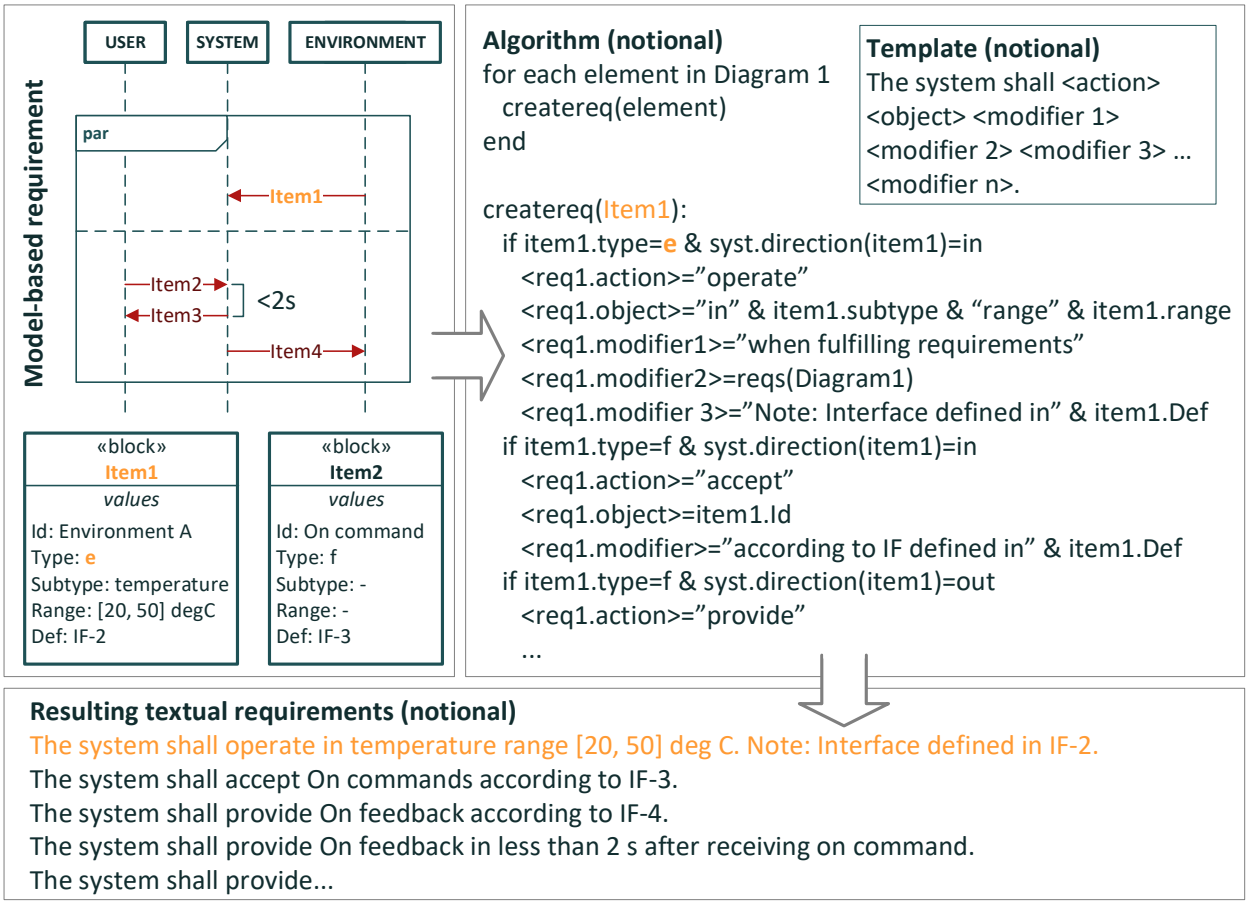


Figure 5. A Representation of the Concept to Generate Textual Requirements Out of Model-Based Requirements

Template

The basic template for a requirement takes the form of *The system shall <action> through <interface>*. This form is refined to capture the richness of requirements offered by the model-based requirements described earlier in the paper. The resulting forms are shown next.

Consider the basic model provided by the sequence diagram in Figure 2 and the internal block diagram in Figure 3. Table 1 shows the template for the requirement in natural language and describes how each element of those model-based requirements is mapped to an element of such template.

Table 1. Mapping of Model Elements to Textual Template

Template of textual requirement	Model element
The <object> shall <accept> <Input> according to <Interface>. *Note 1*: <Input> is defined in <Source 1>. *Note 2*: <Interface> is defined in <Source 2>.	<object>: Block in diagrams referred to as *System*. <accept>: Captured as an input directional port on the system in the Sequence Diagram (incoming arrow in the sequence diagram). <Input>: Name of the *Signal* connected to the input directional port in the Sequence Diagram. <Interface>: Connection between *System* block and external block in the Internal Block Diagram, to which *Signal* is allocated. This is described as a physical port in the System block. <Source 1>: Properties of the *Signal*, directly described in the properties of the element. <Source 2>: Properties of the physical interface, directly described in the properties of the *Port* element.
The <object> shall <provide> <Output> according to <Interface>. *Note 1*: <Output> is defined in <Source 1>. *Note 2*: <Interface> is defined in <Source 2>.	<object>: Block in diagrams referred to as *System*. <provide>: Captured as an output directional port on the system in the Sequence Diagram (outgoing arrow in the sequence diagram). <Output>: Name of the *Signal* connected to the output directional port in the Sequence Diagram. <Interface>: Connection between *System* block and external block in the Internal Block Diagram, to which *Signal* is allocated. This is described as a physical port in the System block. <Source 1>: Properties of the *Signal*, directly described in the properties of the element. <Source 2>: Properties of the physical interface, directly described in the properties of the *Port* element.

Consider now the model-based requirements in Figure 6, which capture required dependencies between the inputs and outputs. It should be noted that the three examples are not exhaustive, but other types of dependencies may be captured (Salado & Wach, 2019). Table 2 shows the templates for the requirement in natural language and describes how each element of model-based requirements is mapped to an element of such templates.

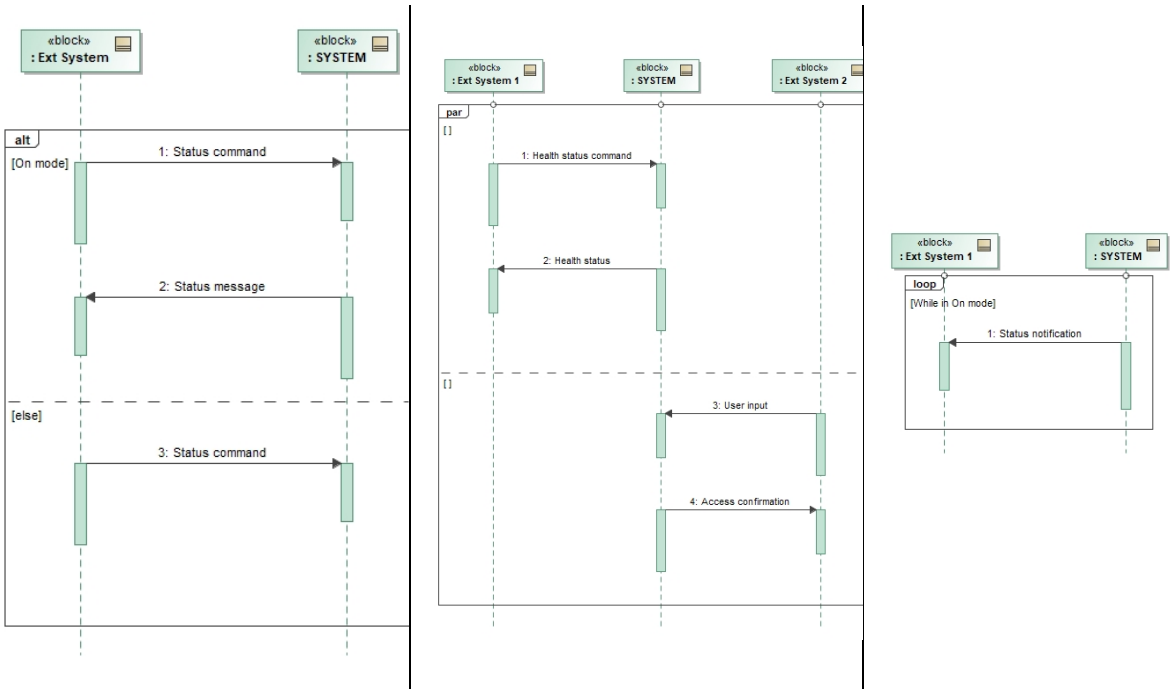


Figure 6. Examples of Model-Based Requirements Capturing Various Dependencies Between Inputs and Outputs

Note. Left: alternative required exchange based on conditions; Center: exchanges that need to be executed in parallel; Right: continuous exchange until a condition is met.

Table 2. Mapping of Functional Dependencies Model Elements to Textual Template

Template of textual requirement	**Model element**
The <object> shall <action> <when> in <condition>.	<object>: Block in diagrams referred to as *System*. <action>: It takes the value of *accept* or *provide* depending on whether the *Signal* element inside one of the branches of the conditional element is an input or an output, respectively to the block *System*. <when>: This value is used when the diagram element is *alt*. <condition>: As described in the condition property of the *alt* element.
The <object> shall <action 1> <while> <action 2>.	<object>: Block in diagrams referred to as *System*. <action 1>: It takes the value of *accept* or *provide* depending on whether the *Signal* element inside one of the branches of the conditional element is an input or an output, respectively to the block *System*. <while>: This value is used when the diagram element is *par*. <action 2>: It takes the value of *accept* or *provide* depending on whether the *Signal* element inside another branch of the conditional element is an input or an output, respectively to the block *System*.
The <object> shall <action> <while/for> <condition>.	<object>: Block in diagrams referred to as *System*. <action>: It takes the value of *accept* or *provide* depending on whether the *Signal* element inside the conditional element is an input or an output, respectively to the block *System*. <while/for>: This value is used when the diagram element is *loop*. <condition>: As described in the condition property of the *alt* element.

It should be noted that defining required time dependencies or restrictions between inputs and outputs may also be necessary (Salado & Wach, 2019). Figure 7 shows an example. In this case, Table 3 shows the template for the requirement in natural language and describes how each element of model-based requirements is mapped to an element of such template.

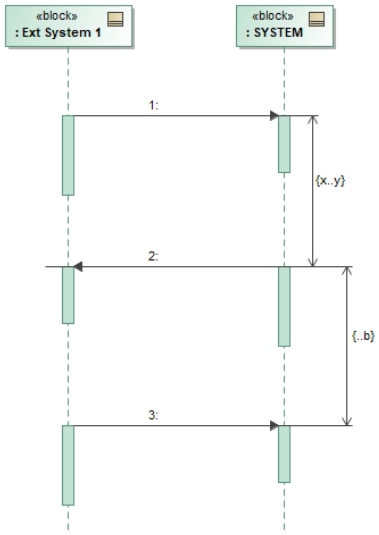


Figure 7. Example of a Model-Based Requirement Capturing Time Restrictions

Table 3. Mapping of Timing Dependencies Model Elements to Textual Template

Template of textual requirement	Model element
The <object> shall <action 1> in <time dependency> <after> <action 2>.	<object>: Block in diagrams referred to as *System*. <action 1>: It takes the value of *accept* or *provide* depending on whether the *Signal* element is an input or an output, respectively to the block *System*. <time dependency>: This is formally defined as a range of [Min, Max], which refer to dependencies such as: less than, more than, within. <after>: This is implied by the temporal dependency given by the *duration constraint*. <action 2>: It takes the value of *receiving* or *providing* depending on whether the *Signal* element is an input or an output, respectively to the block *System*.

Two options are offered for the template for capturing simultaneity of requirement applicability in natural language (as modeled for example in Figure 4). The first one is shown in Table 4, together with a description of how each element of model-based requirements is mapped to an element of such template. The second one consists in simply creating separate sections of the requirement document for each mode requirement, with a statement that reads, *All requirements in this section shall be fulfilled simultaneously*.

Table 4. Mapping of Applicability Simultaneity Model Elements to Textual Template

Template of textual requirement	Model element
<Req X>. The system shall… Note: This requirement must be fulfilled simultaneously with [<Req Y>].	<Req X> is a requirement originating from a *Sequence Diagram* linked to a *state* element. [<Req Y>] is a list of all requirements originating from all *Sequence Diagrams* linked to the *state* element to which *Sequence Diagram* from which <Req X> originates is also connected.

No template is prescribed for capturing the characteristics of inputs, outputs, and interfaces in textual form. In general, they may be listed as columns containing the property and the required values for each property. For physical interfaces, properties may be organized, for example, following a layered approach, such as identifying a transport layer and a physical layer.

Application Example

Case Design

The proposed template to transform the model-based requirements developed in this research project into natural language requirements that can be used to support contractual activities is applied to the case developed in Salado and Wach (2019). In such work, a notional set of requirements in textual form (not necessarily following any template) was transformed into a set of model-based requirements. In this paper, the resulting model-based requirements in such work are transformed back into textual requirements, but using the template presented in this paper. The resulting textual requirements are compared against those used as source requirements in the original work.

It should be noted that a formal comparison of the efficiency, coverage, and accuracy of the resulting requirements after applying the template presented in this paper is outside of the scope of this paper. The focus of the paper is to illustrate how the proposed template can be used to transform model-based requirements to textual requirements, without assessing its performance.

Problem Statement: Model-Based Requirements

The model-based requirements used in this case are depicted in Figures 8 through 15 and directly taken from Salado and Wach (2019). They represent the requirements for an optical space instrument with the purpose to take images of the Earth and send them to the satellite platform under command by the platform. In parallel, the instrument is required to provide health status data *continuously* to the satellite platform for monitoring purposes. The requirement set, which has been adapted from Salado and Nilchiani (2014) and includes new requirements that were added for coherence and partial completeness, provide nevertheless a limited set of requirements with respect to a real-life project. However, the

> acceptability and suitability of the sample requirements [were] validated by deriving and contrasting them against requirements of actual operational and scientific optical space systems developed by different manufacturers for different customers and with a similar level of complexity, which is represented by an instrument mass of around 1 ton. (Salado & Nilchiani, 2014)

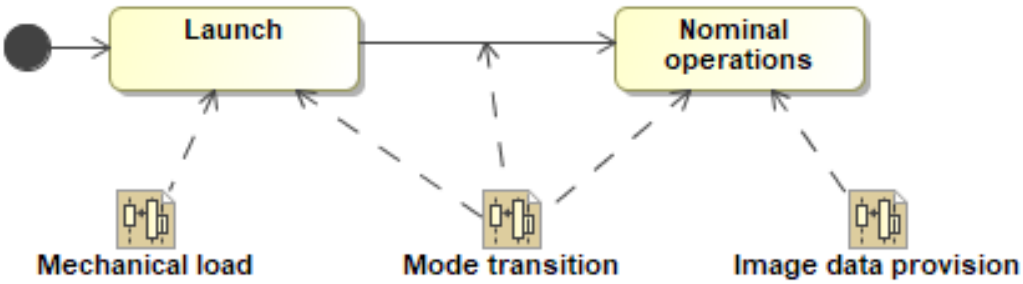


Figure 8. Mode Requirements
(Salado & Wach, 2019)

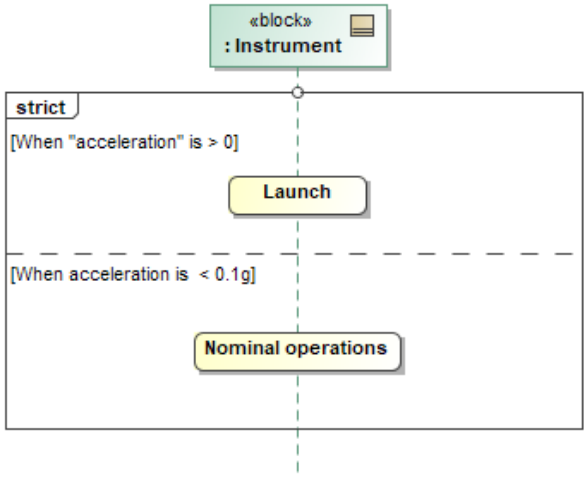


Figure 9. Conditions for Applicability of Each Subset of Requirements (Mode Transition in Figure 8)
(Salado & Wach, 2019)

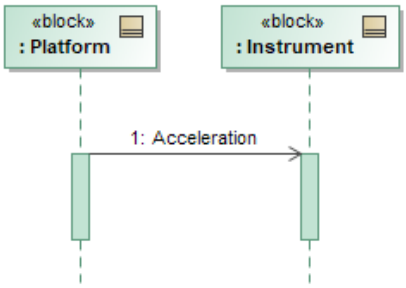
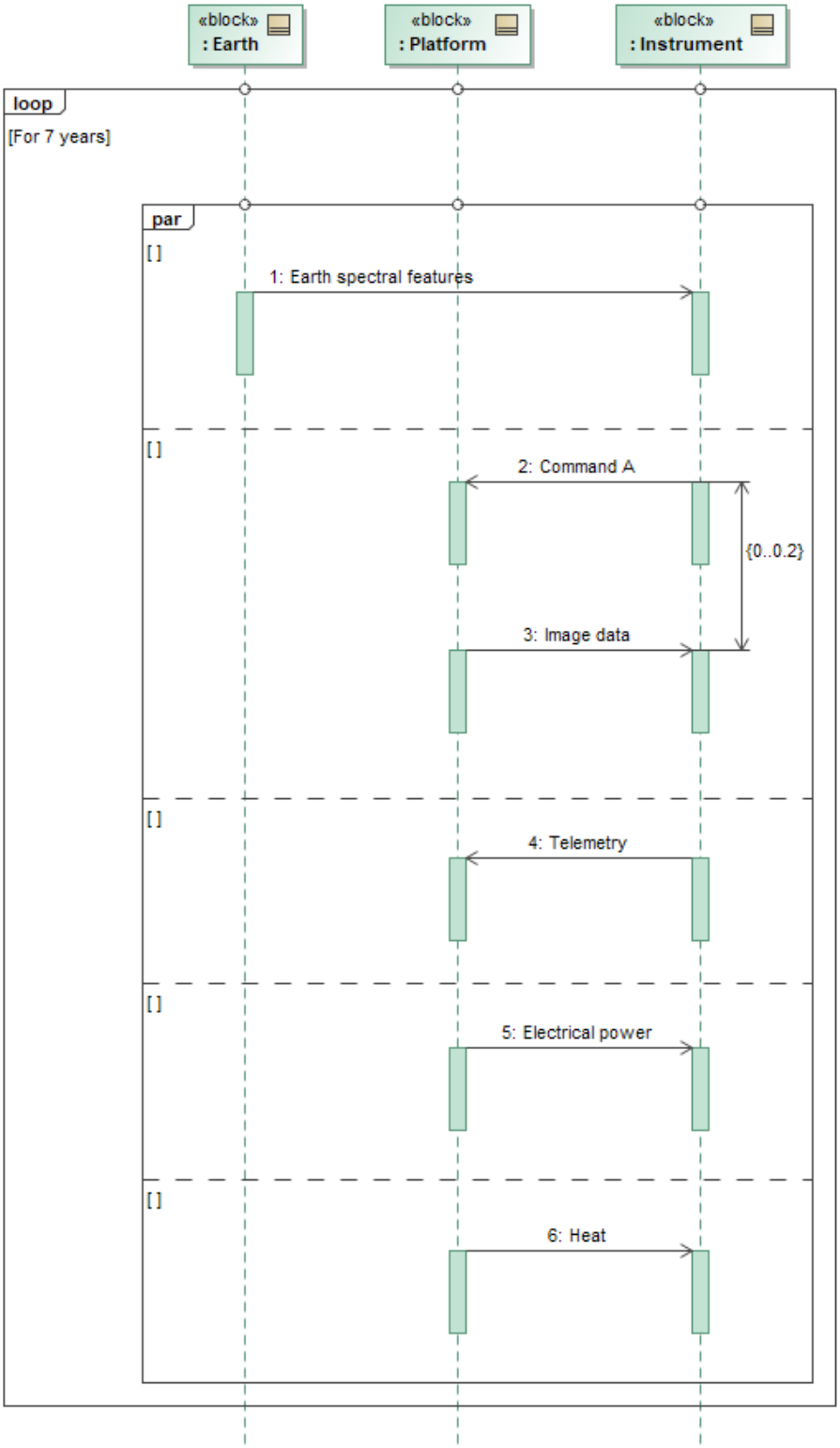


Figure 10. Exchange Related to the Mechanical Load Requirement
(Salado & Wach, 2019)



Note. There is an error in the figure: Command A is an input to the Instrument, and Image data is an output of the Instrument.

Figure 11. Required Exchanges in Nominal Operations
(Salado & Wach, 2019)

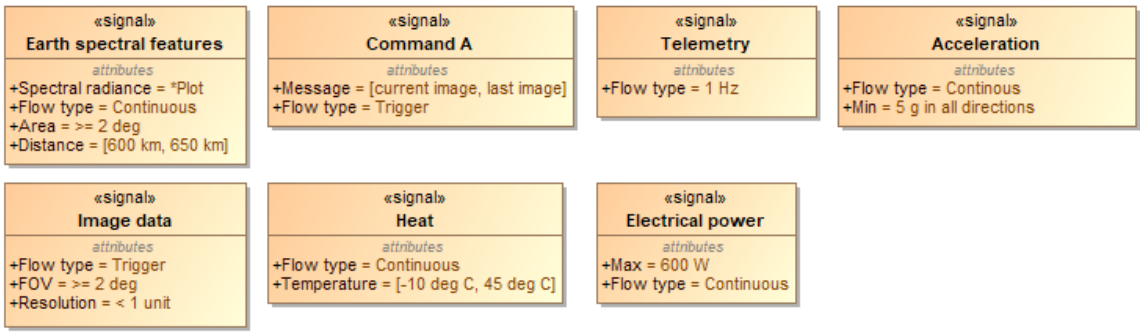


Figure 12. Required Characteristics of the Required Inputs and Outputs
(Salado & Wach, 2019)

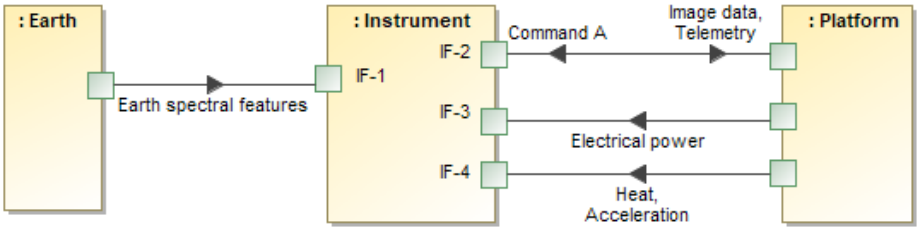


Figure 13. Requirements on the Allocation of Logical Inputs and Outputs to Physical Interfaces Through Which They Must Be Conveyed
(Salado & Wach, 2019)

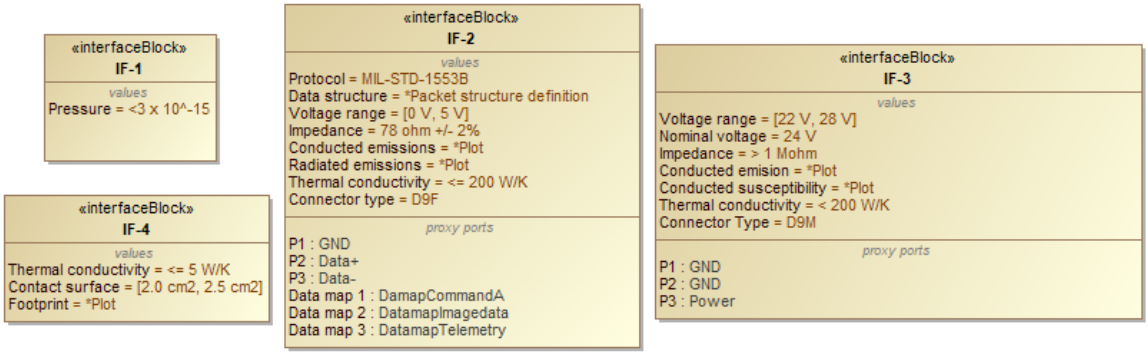


Figure 14. Required Characteristics of the Physical Interfaces Through Which Inputs and Outputs Must Be Conveyed
(Salado & Wach, 2019)

Figure 15. Modeling of Transport Layer Aspects as Proxy Ports for Leveraging Model Complexity
(Salado & Wach, 2019)

Resulting Contractual Requirements in Natural Language

Application of Step 1. Each interface block in Figure 14 is converted to a table form with two columns, one listing the property and one listing the corresponding value. It should be noted that, as part of those properties, the information in Figure 15 is nested for some of the interfaces in Figure 14**.** The resulting tables are not shown in this paper because of length limitations. For referencing purposes in other requirements, they will be referred to as Tables E1 through E4, which correspond to IF-1 through IF-4, respectively.

Application of Step 2. For simplicity, the approach to divide the requirement set in sections is used. Two sections are therefore created. Section 1 corresponds to *Launch* requirements, and Section 2 corresponds to *Nominal Operations* requirements.

Application of Step 3. First, all signals in Figure 12 are converted to a table form with two columns, one listing the property and one listing the corresponding value (ref. Table 5). A template in Table 1 is applied to Figures 10 and 13, yielding a single requirement for the *Launch* requirements subset. All templates are then used on Figures 11 and 13 to generate the requirements for the *Nominal Operations* subset. The resulting requirements are given in Table 6. Requirements R2 through R7 are generated using template in Table 1. Requirement R8 is generated using templates in Table 3. Requirements R9 and R10 are generated using templates in Table 2. Note that R9 and R10 have been simplified because of paper length limitations. Essentially, the requirements should be extended to every action that is paralleled and every action that is part of the lifetime loop, respectively.

Table 5. Required Characteristics of Inputs and Outputs

Property	Value
S1	
Flow type	Continuous
Min	5g in all directions
S2	
Spectral radiance	*Plot
Flow type	Continuous
Area	>= 2 deg
Distance	[600 km, 650 km]
S3	
Message	[current image, last image]
Flow type	Trigger
S4	
Flow type	Continuous
Temperature	[-10 deg C, 45 deg C]
S5	
Max	600 W
Flow type	Continuous
S6	
Flow type	Trigger
Field of View	>= 2 deg
Resolution	< 1 unit
S7	
Flow type	1 Hz

Table 6. Resulting Textual Requirements

ID	Requirement
Launch Note: All requirements in this section must be fulfilled simultaneously.	
R1	The system shall accept Acceleration according to IF-4. Note 1: Acceleration is defined in Table S1. Note 2: IF-4 is defined in Table E4.
Nominal Operations Note: All requirements in this section must be fulfilled simultaneously.	
R2	The system shall accept Earth spectral features according to IF-1. Note 1: Earth spectral features are defined in Table S2. Note 2: IF-1 is defined in Table E1.
R3	The system shall accept Command A according to IF-2. Note 1: Earth spectral features are defined in Table S3. Note 2: IF-2 is defined in Table E2.
R4	The system shall accept Electrical power according to IF-3. Note 1: Earth spectral features are defined in Table S4. Note 2: IF-3 is defined in Table E3.
R5	The system shall accept Heat according to IF-4. Note 1: Earth spectral features are defined in Table S5. Note 2: IF-4 is defined in Table E4.
R6	The system shall provide Image data according to IF-2. Note 1: Earth spectral features are defined in Table S6. Note 2: IF-2 is defined in Table E2.
R7	The system shall accept Telemetry according to IF-2. Note 1: Earth spectral features are defined in Table S7. Note 2: IF-2 is defined in Table E2.
R8	The system shall provide Image data in less than 0.2 s after having received Command A.
R9	The system shall accept Earth spectral features while accepting [Command A, Electrical Power, Heat] and providing [Image data, Telemetry].
R10	The system shall <all actions> for 7 years.

Step 4 has not been applied in this example.

Comparison and Discussion

The resulting textual requirements for the required properties of the physical interfaces captured in Figures 14 and 15 are identical to those in the benchmark given in Salado and Wach (2019), although they have not been explicitly shown in this paper. However, a comparison of the description of the resulting requirements in table form with those tables in the source paper yield this conclusion.

With respect to requirements in Tables 5 and 6, it is necessary to look at the benchmark textual requirements, which are listed in Table 7 directly from the original source in Salado and Wach (2019). The requirement sets look different at first sight and, in fact, present also some differences with respect to the solution space. They are discussed next.

Table 7. Benchmark Textual Requirements

(Adapted from Salado & Wach, 2019)

Req ID	Description
BR1	The instrument shall image a target at 600 km–650 km according to IF-1.
BR2	The instrument shall image a target with spectral radiance of ABC (*plot) according to IF-1.
BR3	The instrument shall accept Command A according to IF-2.
BR4	The instrument shall transmit image data according to IF-2 in less than 0.2 s after receiving Command A.
BR5	The instrument shall have a resolution better than 1 unit.
BR6	The instrument shall have a FOV greater than 2°.
BR7	The instrument shall provide telemetry data every 1 s according to IF-2.
BR8	The instrument shall accept power according to IF-3.
BR9	The instrument shall consume less than 600 W of electrical power.
BR10	The instrument shall withstand a mechanical load of 5 g in any direction on IF-4.
BR11	The instrument shall fulfill its performance when subjected to a temperature between -10 deg C and +45 deg C at IF-4.
BR12	The instrument shall have a lifetime of at least 7 years.
Note 1	R10 only applies during launch. All other requirements only apply once the instrument is powered on through IF-3.

In terms of visual differences, a different approach is taken for describing the different modes. However, this is purely a stylistic matter and of no real concern for the definition of the solution space. In addition, the benchmark employed a single requirement for each required property of the required system inputs and outputs, whereas the resulting set in this paper employs a table form for the properties linked to a single requirement for each input and output. We believe that both options have pros and cons with respect to requirement management. For example, the benchmark option may be easier to manage in terms of traceability in requirements management tools. However, it does not present any structure to facilitate consistency during requirement elicitation. Certainly, there may be

ways to overcome both problems with both approaches. Hence, these differences remain aesthetic and with no impact on the definition of the solution space. Therefore, they can be considered equivalent.

Wording employed in the textual statements is also different. The free form employed in the benchmark yields the use of verbs that provide a description of the intent or purpose expected to be fulfilled by the system, whereas the proposed template uses only accepting/providing statements. We argue that the proposed approach is actually more effective. We base this assertion on two aspects. First, the purpose of deriving stakeholder needs into system requirements is to devoid the requirements of context, so that only what the system has to do is defined, not what an external actor will do with the actions of the system. In this sense, and using systems theory, a system can be fully characterized by the inputs it accepts from the environment and external systems and the output it provides to them. Second, natural language lends itself towards diversity of interpretation. This difference can cause a difference in the content of the solution space, as different engineers work towards finding an acceptable solution. Therefore, limiting the types of actions that the system can take, as proposed in this paper, may be beneficial to cope with such limitation of natural language.

In terms of effects on the solution space beyond wording interpretation, the only apparent difference is that the benchmark did not explicitly refer to the need to execute certain actions in parallel, while the models did. We believe that this difference is just an artifact of the limitations of the case study but felt it was necessary to mention for completeness. Therefore, we consider both sets of requirements to be equivalent from this perspective.

Finally, it should be noted that the transition requirements captured in Figure 9 have not been transformed to textual requirements. The reason is that the model-based requirements were incomplete and did not capture the external conditions for the different mode requirements as external inputs (particularly, pressure conditions), but just as operational conditions of the transitions. Because of this lack of completeness, the templates cannot be applied in this case.

Conclusions

Prior work in the frame of this research project demonstrated an approach to capture requirements directly in model-based form without using requirements in natural language, such as the traditional *shall* requirement statements. This paper has shown a template to generate contractual requirements in natural language directly out of those model-based requirements. These templates can enable a technical team to transition to model-based requirements while guaranteeing fulfillment of the expectation of contractual departments and acquisition programs. The former can work directly in developing models, while the latter can still provide *shall* statements to vendors and suppliers.

It should be noted that the effort is ongoing and is planned to be completed within the timeframe of the NPS Research Acquisition Program's “Automatic Generation of Contractual Requirements from MBSE Artifacts” project.

References

Buede, D. M. (2009). *The engineering design of systems: Models and methods*. Wiley.

Dada, D. (2006). The failure of e-government in developing countries: A literature review. *Electroninc Journal of Information Systems in Developing Countries, 26*, 1–10.

El Eman, K., & Birk, A. (2000). Validating the ISO/IEC 15504 measure of software requirements analysis process capability. *IEEE Transactions in Software Engineering, 26*, 541566.

Friedenthal, S., Moore, A., & Steiner, R. (2015). *A practical guide to SysML—The systems modeling language.* Waltham, MA: Morgan Kaufman.

Holt, J., Perry, S., Payne, R., Bryans, J., Hallerstede, S., & Hansen, F. O. (2015). A model-based approach for requirements engineering for systems of systems. *IEEE Systems Journal, 9*, 252–262.

Holt, J., Perry, S. A., & Brownsword, M. (2011). *Model-based requirements engineering*. IET.

INCOSE. (2012). *Guide for writing requirements*. The International Council of Systems Engineering.

INCOSE. (2015). *Systems engineering handbook: A guide for system life cycle processes and activities.* Hoboken, NJ: John Wiley and Sons.

Kossiakoff, A., Sweet, W. N., Seymour, S. J., & Biemer, S. M. (2011). *Systems engineering principles and practice.* Hoboken, NJ: John Wiley & Sons.

McConnell, S. (2001). From the editor—An ounce of prevention. *IEEE Software, 18*, 5–7.

Micouin, P. (2008). Toward a property based requirements theory: System requirements structured as a semilattice. *Systems Engineering, 11*(3).

Miotto, B. L. A. P. (2014). Model-based requirement generation. 2014 IEEE Aerospace Conference, Big Sky, MT.

Salado, A., & Nilchiani, R. (2014). A categorization model of requirements based on Max-Neef's model of human needs. *Systems Engineering, 17*, 348–360.

Salado, A., & Nilchiani, R. (2017). Reducing excess requirements through orthogonal categorizations during problem formulation: Results of a factorial experiment. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 47*, 405–415.

Salado, A., Nilchiani, R., & Verma, D. (2017). A contribution to the scientific foundations of systems engineering: Solution spaces and requirements. *Journal of Systems Science and Systems Engineering, 26*, 549–589.

Salado, A., & Wach, P. (2019). Constructing true model-based requirements in SysML. *Systems, 7*, 19.

Von Bertalanffy, L. (1969). *General systems theory—Foundations, development, applications.* New York, NY: George Braziller.

Wymore, A. W. (1993). *Model-based systems engineering.* Boca Raton, FL: CRC Press.

Yeo, K. T. (2002). Critical failure factors in information system projects. *International Journal of Project Management, 20*, 241–246.

Acknowledgements & Disclaimer

This material is based upon work supported by the Acquisition Research Program under HQ0034-18-1-0006. The views expressed in written materials or publications, and/or made by speakers, moderators, and presenters, do not necessarily reflect the official policies of the Department of Defense nor does mention of trade names, commercial practices, or organizations imply endorsement by the U.S. Government.

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL
555 DYER ROAD, INGERSOLL HALL
MONTEREY, CA 93943

www.acquisitionresearch.net